# Home Insurance Management System

## Technology stack

1. SQL Server for database
2. Entity framework core for data access
3. .Net core MVC for presentation
4. .Net core Hosted service for scheduled tasks
5. Memory cache for caching
6. JustMock for mocking
7. Microsoft DI container for dependency injection

## Solution structure

Solution has the following layers. Currently these are under single project. Can be broken down to separate projects if needed.

### Infrastructure:

All cross cutting concerns should be in this layer. Eg: caching.

### Models:

All models including entities, view models and dto are here. Entities should be maintained as POCO and configurations should be done through fluent api to ensure that this layer is portable

### Repository:

This is a wrapper around entity framework core. All data should be accessed through repository. This will help us to encapsulate entity framework to single layer, maintain consistency in data handling and consistency in exception handling.

### Domain services:

All business logic is in this layer. This layer will access data from repository, perform business operations, convert entities to view models and return to presentation. We can use libraries like automapper to simplify object mapping.

### Unit tests

All unit tests are in this layer, currently grouped under respective layers. If layers are separated to individual projects, this also should be separated.

Builder pattern is used to build test data.

## Notes

1. Entire solution is built in DI pattern. All classes can be built and unit tested independently.
2. JustMock is used to mock dependencies and unit test classes. Fully executable unit tests are available in unittests project.
3. HostedService (IHostedService & BackgroundService) is used to automatically renew insurance. Template implementation is available in AutomaticPolicyRenewalService.
4. I was not sure what exactly we mean by “setting up insurance policies for insurance companies”. Instead of this, I have designed a quote module. Based on property attribute (area, zip code etc), insurance companies will provide a quote. Borrower can review quotes and buy insurance.
5. Exception handling and logging are left at defaults provided by .Net core. Custom middleware can be added if more sophisticated handling is required.
6. Authentication is not implemented yet. We can use windows authentication as most of the tasks are done by internal users. If more sophisticated control is required, we can use asp.net identity framework to implement register/ login/ user management and authorization.
7. BankController, corresponding services & repositories are completely implemented. We can view/ add/ edit/ delete banks.
8. A client side framework such as knockout.js or jQuery & bootstrap can be used to enhance user experience.

## Assumptions

1. No integration with external systems
2. Manage banks/ insurance company/ property/ loan/ borrower is an admin task and will be done by an internal user
3. Borrower can come to the site, view insurance quotes and purchase insurance